(NeXT Tip #16) Programming NXDefaults

Christopher Lane (*lane[at]CAMIS.Stanford.EDU*)
Mon, 25 Jan 1993 14:55:29 -0800 (PST)

NeXT applications have a concept of 'defaults' which can be modified, saved and reloaded on subsequent application runs. These defaults are user preferences, window positions, color choices, etc. However, NeXT's API for defaults from a programmer's point of view, is suboptimal -- you'll find very few NeXT-supplied code examples that use defaults in a non-trivial way.

What you will find, in documentation about the defaults system and in code you get off the InterNet, are hard-coded vectors of defaults in source code:

```
static NXDefaultsVector myDefaults = {
{ "hsize", "8.5" },
{ "vsize", "11" },
{ "reversed", "true" },
{ "compressed", "false" },
...
{ "defaultformat", "tex" },
{ NULL, NULL }
};
```

This is bad for several reasons -- including the style issue of putting constants in the body of a program. Users can override these parameters via the application's preferences panel or a Unix-level 'dwrite', system administrators who want to correct a default for all users, e.g. NewsGrazer's default 'NewsHost', have to modify and recompile the application if source is available or in NewsGrazer's case where it isn't, remind everyone that the defaults have to be modified to specific local values before the program will run. There's also no way to know what are all the defaults are in a program without looking through the source code or examining the binary under gdb.

Manipulating NeXT defaults is also clumsy as NeXT only provides the ability to read and write strings, other constructs like booleans, numbers, points, colors etc. you have to build up for yourself:

```
#define getFloatDefault(s) atof(NXGetDefaultValue([NXApp appName], s))
```

After building a number of applications using defaults, when developing the 'Tuner' application at home, I decided there had to be a better way. Rather than try to write the 'perfect' solution, I decided to leverage one of NeXT's own AppKit classes (NXStringTable) to simplify things a little bit.

The result is available on ~lane/Programming/DefaultsTable and consists of a subclass of NXString called a DefaultsTable and a number of C macros and routines to make handling defaults easier. The primary feature of this scheme is that defaults are no longer defined in your code but rather in a file in your *.app application bundle. A typical defaults file (which I generally call 'Defaults.strings' but you can name as you please):

```
"AutoUpdate" = "No";
"Hosts" = "";
"QueueLength" = "All";
"ShiftDisplay" = "0.333";
"BackgroundColor" = "r:0.6667 g:0.6667 b:0.6667 a:-1";
"Origin" = "x:0.0 y:0.0";
"MaximumSize" = "w:100 h:100";
```

This is the standard NeXT NXStringFile format. (Usually used for language translation). The defaults themselves are not known to the application until runtime and the Defaults.strings file in the *.app directory can be edited as needed at installation time, without the program's source code. You can also

include C-style (/* ... */) comments in the file. This is not an alternative
to the NeXT's defaults system, just a different API for it.

The DefaultsTable object only implements four additional methods (to those the
NXStringTable object provides):

- initFromFile:(const char *) file;
- registerDefaults:(const char *) owner;
- writeDefaults:(const char *) owner;
- updateDefaults;

It also defines the following C macros & functions:

getDefault(const char *name) aka getStringDefault()
writeDefault(const char *name, const char *value) aka writeStringDefault()
getIntDefault(const char *name)
writeIntDefault(const char *name, int value)
BOOL getBoolDefault(const char *name)
writeBoolDefault(const char *name, (BOOL) value)
float getFloatDefault(const char *name)
writeFloatDefault(const char *name, float value)
NXSize getSizeDefault(const char *name);
writeSizeDefault(const char *name, NXSize size)
NXPoint getPointDefault(const char *name);
writePointDefault(const char *name, NXPoint point)
NXColor getColorDefault(const char *name);
writeColorDefault(const char *name, NXColor color)

For composites like NXPoint, NXSize, NXColor data structures, it writes and
reads formatted strings, e.g. "x:230 y:450" to make understanding and editing
the defaults in the Defaults.strings file simpler. For examples of use of
this object class, see the following subdirectories of ~lane/Programming:

ArchiveBrowser, MOTD, Magnify, Unknown & NLoad

These all had the DefaultsTable class retrofitted -- which isn't hard to do.
In some of these files, you will also see examples of using the 'tag' slot of
interface Control subclasses as a boolean to flag which defaults have been
changed by the user and need to be saved. (Not related to DefaultsTable.)

You can copy (or link) the DefaultsTable.* files to your development
directory, add the class via ProjectBuilder, add the Defaults.strings file as
an 'Other Resources' item via ProjectBuilder, and #import the DefaultsTable.h
file as needed. This package is 'work in progress' -- I've generally included
routines that I needed for the applications I've used it with. I'm happy to
extend it based on the needs of other's who want to use it.

- Christopher

PS: To get off this list, send EMail to KSL-NeXT-Request[at]CAMIS.Stanford.EDU